Gaussian Mixture Belief Space Reinforcement Learning

A Thesis
Presented to
the Academic Faculty

by

Gabriel Nakajima An

In Partial Fulfillment of the
Requirements for the Degree of
Bachelor of Science in Computer Science
at the College of Computing

Georgia Institute of Technology
December 2018

Gaussian Mixture Belief Space Reinforcement Learning

Approved by:

Professor Evangelos A. Theodorou, *Advisor*
School of Aerospace Engineering
Georgia Institute of Technology

Professor Byron Boots
School of Computer Science
Georgia Institute of Technology

Date Approved: December 12, 2018

To my family: Mãe, Pai, Vini, e Davi.

Acknowledgements

I would like to thank a number of people who were of great importance to me during this thesis work.

First and foremost, I would like show gratitude to my advisor Professor Evangelos Theodorou, who guided me throughout the entire process and taught me invaluable lessons. I am thankful for my other mentor Dr. Yunpeng Pan, who also provided precious guidance throughout this thesis work and who introduced me into the ACDS group.

I owe immense gratitude to Professor Byron Boots, who introduced me to the field of machine learning and played a great role in the start of my research career.

I am immensely grateful to have incredible labmates, with whom I shared countless meaningful discussions: Dr. Kaivalya Bakshi, Marcus Pereira, George Boutselis, David Fan, Keuntaek Lee, Harleen Brar, Pat Wang, Manan Gandhi, Ethan Evans, Grady Williams, Bogdan Vlahov, and Sam Ting. I would like to thank my friend, Yousef Emam, who is not only an incredible researcher, but also one of my best friends. I am thankful for students from other labs: Ching-An Cheng, Xinyan Yan, Nolan Wagener, Mustafa Mukadam, Jake Sacks, Stella Kampezidou, and Chams Eddine Mballo.

Finally, I would like to thank my family.

Abstract

In reinforcement learning and optimal control, one successful approach to address system stochasticity and epistemic uncertainty in the dynamics model is to consider, rather than a single state, a distribution over the states, i.e. a *belief*. This belief is usually described using a Gaussian distribution; however, this representation may be very limited due its significant approximation error when describing multimodal distributions. This thesis addresses this problem by representing the belief with a mixture of Gaussians and presenting a reinforcement learning algorithm that optimizes a sequence of controls with this particular belief space representation. We show that this method can propagate belief much more accurately and we apply this algorithm to trajectory tracking tasks with stochastic dynamics.

Contents

Acknowledgements **III**

Abstract **III**

1 Introduction **1**

2 Background Theory **4**

2.1 Reinforcement Learning & Optimal Control 4

2.1.1 Problem Formulation 4

2.1.2 Dynamic Programming 5

2.1.3 Differential Dynamic Programming 6

2.2 Model Learning with Gaussian Processes 12

2.2.1 Bayesian Linear Regression 12

2.2.2 Gaussian Processes 13

2.3 Probabilistic Differential Dynamic Programming 15

2.3.1 Gaussian Belief One-Step Propagation via Moment Matching 16

2.3.2 Belief cost 19

3 Methods **20**

3.1 Gaussian Mixture Splitting 20

3.1.1 Criterion for Splitting 21

3.2 Gaussian Mixture Belief Reinforcement Learning 23

4 Results **26**

4.1 Gaussian Splitting 26

4.2 Moment matching 26

5 Conclusion **29**

5.1 Future Work 30

References **31**

Chapter 1

Introduction

The field of machine learning is concerned with extracting information from data and representing it in some efficient manner in order to accomplish something useful. As a subfield of machine learning, *reinforcement learning* (RL) has the objective to interact with a *world* and learn how to make the "best" decisions to achieve a specific goal.

The following diagram (Fig. 1.1) shows a pictorial representation of the problem setup. At each instant in time, an *agent* interacts with a *system* by applying a *control*. Resultingly, the *state* of the system changes and the agent perceives this change through sensory inputs. In addition, the system incurs a specific *cost* value (penalty) on the agent. The objective of the RL problem, therefore, is to apply controls to the system such that the agent is incurred the least amount of cost over a period of time. As a result, the goal that the agent must accomplish is encoded in the form of the cost.

In the original formulation of RL, the agent originally has no knowledge on how the system evolves with the control inputs over time, nor does it know the mechanism behind how the system incurs the cost. However, in the field of control, RL is casted as a problem named *optimal control*[Bertsekas, 2000]. Here, the agent has full knowledge of how the system behaves and how the cost is incurred. More precisely, the behavior of the system is described by how its state evolves over time as a function of the applied controls. This behavior is represented in the form of a *dynamics* function, which maps the control and current state to the corresponding

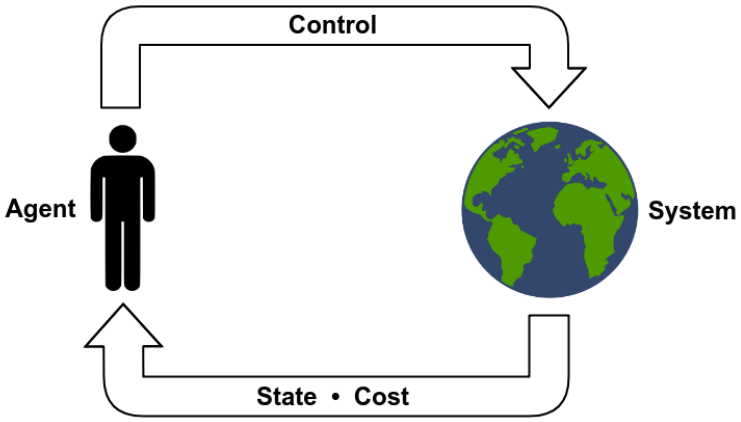

Figure 1.1: RL setup.

change in state. Additionally, the cost is characterized in the form of a *cost function*, which maps the control and current state to a real-value cost.

In this thesis, we consider the RL problem in which the form of the cost function is known, while the dynamics is not. This formulation is, in fact, quite general. In most practical RL problems, such as in robotics, the form of the cost function is known and generally hand-designed, as the goal of the task is already defined. Therefore, for practical considerations, it is a reasonable assumption to assume knowledge on the cost function. However, we assume unknown dynamics because most systems are extremely hard to model, and it is, therefore, unreasonable to assume full knowledge on how the system evolves over time.

In addition, we consider a formulation wherein the system is not *fully observable*, that is, there exists uncertainty in how the agent perceives the system due to noisy or partial measurements of the state. As a result, we cannot know the true state of the system, but rather, we must represent this uncertainty in the form of a probability distribution over the state space, defined as the *belief* of the state.

The most common approach to control a system in *belief space* is by representing the belief with a Gaussian distribution at each instant in time [Pan and Theodorou, 2014, Deisenroth and Rasmussen, 2011, Pan et al., 2015, Deisenroth et al., 2015]. However, the problem with these methods is that representing the true state belief with a Gaussian can be extremely erroneous. For instance, the approximation error of a strongly bimodal distribution with a Gaussian is

significantly high. One way to address this issue is by representing the belief with discrete particles or samples of the state space [Platt et al., 2017, 2012]. This approach, however, is not scalable. It suffers from the *curse of dimensionality*, as the number of particles that the method requires grows exponentially in the number of the dimensions of the state space.

The motivation of this thesis is, therefore, to address the problem of representing the state belief in an efficient, yet general manner, and ultimately learn to control a system in belief space. We do this by proposing an algorithm named Gaussian Mixture Differential Dynamic Programming (GaMix-DDP), which extends a belief space model-based RL algorithm called Probabilistic Differential Dynamic Programming (PDDP) [Pan and Theodorou, 2014]. In GaMix-DDP, the belief is parameterized with a Gaussian mixture with variable number of modes. This belief space representation is general enough to be able to represent multimodal distributions, and it is represented and incorporated into the algorithm in a computationally efficient manner. This efficiency results from using a Bayesian non-parametric method, namely Gaussian process (GP), to learn the system dynamics. Using GPs allows the algorithm to not only be computationally efficient, but also data-efficient, as will be later shown. Finally, we show that GaMix-DDP can plan highly accurate trajectories in belief space and optimize these trajectories very efficiently.

Chapter 2

Background Theory

In this chapter, we provide the technical background of which this work is theoretically founded upon. In section 2.1, we formalize the problem of RL and optimal control, we describe the optimality principle, and we detail an approach to solving optimal control problems. Next, in section 2.2, we describe a Bayesian non-parametric method for regression and inference.

2.1 Reinforcement Learning & Optimal Control

Optimal control and RL were separately developed; however, they essentially aim to solve the same problem. The subtle difference, as discussed in chapter 1, is that optimal control has the extra assumption of knowing the form of the dynamics. Next, we mathematically formulate the optimal control and RL problem.

2.1.1 Problem Formulation

We consider continuous-time dynamics described by the following differential equation

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = f(\mathbf{x}, \mathbf{u}) \ , \tag{2.1}$$

where $\mathbf{x} \in \mathbb{R}^d$ is the state, $\mathbf{u} \in \mathbb{R}^p$ is the control, and $f : \mathbb{R}^d \times \mathbb{R}^p \to \mathbb{R}^d$ describes the rate of change of the state, given the current state and control applied. The objective is to find a control *policy* $\pi : \mathbb{R}^d \times \mathbb{R} \to \mathbb{R}^p$ such that the accumulated cost over a finite-horizon period is minimized, subject to dynamics constraints. That is, find $\pi^* = \arg\min_\pi J\left(\mathbf{x}(t_0)\right)$, where

$$J\left(\mathbf{x}(t_0)\right) = \Phi\left(\mathbf{x}(T)\right) + \int_{t_0}^{T} L\left(\mathbf{x}(t), \pi(\mathbf{x}(t), t)\right) \mathrm{d}t \ . \tag{2.2}$$

In Eq. 2.2, t_0 is the initial time of optimization, T is the horizon or final time, $\Phi : \mathbb{R}^d \to \mathbb{R}$ is the terminal cost which incurs cost only at the terminal state $\mathbf{x}(T)$, and L is the Lagrangian (term borrowed from quantum mechanics), which describes the running (or instantaneous) cost. Additionally, notice that we described the state as a function of time, since the state evolves with time according to the dynamics and the controls applied through the policy π.

2.1.2 Dynamic Programming

We introduce the *value function* or *cost-to-go*, which is an essential notion to the framework of optimal control and RL. The value function V is defined as a mapping from an initial state and time to the optimal value of the objective J:

$$V(\mathbf{x}(t_0), t_0) = \min_\pi \left[\Phi\left(\mathbf{x}(T)\right) + \int_{t_0}^{T} L(\mathbf{x}, \pi(\mathbf{x}, t)) \, \mathrm{d}t \right] \tag{2.3}$$

Using the Bellman principle of dynamic programming [Bellman, 1957], we will express the value function recursively. For this, we discretize time using the Euler scheme by indexing time with $k = 1, 2, \ldots, H$ and time intervals of size $\delta t = \frac{T - t_0}{H - 1}$, so we have that $t_k = t_0 + k\delta t$. We also simplify the notation by subscripting the time index, e.g. $\mathbf{x}_k = \mathbf{x}(t_k)$. We can express the discretized dynamics as

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \int_{t_k}^{t_{k+1}} f(\mathbf{x}, \mathbf{u}) \, \mathrm{d}t \tag{2.4}$$

$$\approx \mathbf{x}_k + \Delta\mathbf{x}_k \ , \tag{2.5}$$

where $\Delta \mathbf{x}_k = f(\mathbf{x}_k, \mathbf{u}_k)\delta t$, and the approximation emerges from the time discretization. Using this discretization, we can apply the Bellman optimality principle and rewrite Eq. 2.3 recursively as

$$V(\mathbf{x}_k, t_k) = \min_{\mathbf{u}(\cdot)} \left[\int_{t_k}^{t_k+\delta t} L(\mathbf{x}, \mathbf{u})\, \mathrm{d}t + V(\mathbf{x}(t_k + \delta t), t_k + \delta t) \right] \tag{2.6}$$

$$= \min_{\mathbf{u}(\cdot)} \left[L(\mathbf{x}_k, \mathbf{u}_k)\delta t + V(\mathbf{x}_{k+1}, t_{k+1}) \right] \tag{2.7}$$

$$= \min_{\mathbf{u}(\cdot)} \left[\mathcal{L}(\mathbf{x}_k, \mathbf{u}_k) + V(\mathbf{x}_{k+1}, t_{k+1}) \right] \ , \tag{2.8}$$

where we define $\mathcal{L}(\mathbf{x}_k, \mathbf{u}_k) \triangleq L(\mathbf{x}_k, \mathbf{u}_k)\delta t$. This form is central to solving optimal control and RL problems. In particular, we will use it to approximate the value function local to a nominal state-control trajectory by starting at the terminal state and propagating the gradient and the hessian of the value function backwards in time. This is explained further in detail in the next section.

2.1.3 Differential Dynamic Programming

We tackle the optimal control problem (Eq. 2.2) by first considering linear dynamics and running cost quadratic in the state and control:

$$\mathbf{x}_{k+1} = \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k \ , \tag{2.9}$$

$$L(\mathbf{x}_k, \mathbf{u}_k) = \mathbf{x}_k^\mathsf{T}\mathbf{Q}\mathbf{x}_k + \mathbf{u}_k^\mathsf{T}\mathbf{R}\mathbf{u}_k \ . \tag{2.10}$$

If the problem is structured as above, then an exact optimal control policy can be found. This is solved using the Linear Quadratic Regulator (LQR) [Jacobson and Mayne, 1970].

To address nonlinear systems and nonquadratic (but still differentiable) cost functions, we can extend LQR by considering an approximation local to a state-control trajectory. We approximate the problem locally by linearizing the dynamics and taking the quadratic expansion of the cost function. This algorithm is called Differential Dynamic Programming (DDP).

Let us now consider a nonlinear, differentiable, discrete-time dynamics,

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) \ . \tag{2.11}$$

Under this dynamics, we generate a nominal state trajectory, $(\bar{\mathbf{x}}_0, \ldots, \bar{\mathbf{x}}_H)$, using predefined nominal control trajectory $(\bar{\mathbf{u}}_0, \ldots, \bar{\mathbf{u}}_H)$. Next, we consider the linear approximation of the discrete dynamics, local to this nominal state trajectory, by taking the first order Taylor expansion of $\mathbf{f}(\mathbf{x}, \mathbf{u})$:

$$\begin{aligned}
\mathbf{x}_{k+1} &= \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) && (2.12)\\
&= \mathbf{f}(\bar{\mathbf{x}}_k + \ \delta\mathbf{x}_k, \bar{\mathbf{u}}_k + \ \delta\mathbf{u}_k) \\
&= \mathbf{f}(\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k) + \mathbf{f}_{\mathbf{x}}^{(k)}\, \delta\mathbf{x}_k + \mathbf{f}_{\mathbf{u}}^{(k)}\, \delta\mathbf{u}_k && (2.13)\\
\Rightarrow\ \delta\mathbf{x}_{k+1} &= \mathbf{f}_{\mathbf{x}}^{(k)}\, \delta\mathbf{x}_k + \mathbf{f}_{\mathbf{u}}^{(k)}\, \delta\mathbf{u}_k \ , && (2.14)
\end{aligned}$$

where $\delta\mathbf{x} \triangleq \mathbf{x} - \bar{\mathbf{x}}$ and analogously for $\delta\mathbf{u}$, and $\mathbf{f}_{\mathbf{x}} \in \mathbb{R}^{d\times d}$ and $\mathbf{f}_{\mathbf{u}} \in \mathbb{R}^{d\times p}$ are the Jacobians of $\mathbf{f}$ with respect to $\mathbf{x} \in \mathbb{R}^d$ and $\mathbf{u} \in \mathbb{R}^p$, respectively. Also, note that the $\mathbf{f}_{\mathbf{x}}$ and $\mathbf{f}_{\mathbf{u}}$ Jacobians are evaluated at $\bar{\mathbf{x}}$ and $\bar{\mathbf{u}}$. To simplify the notation, any function not explicitly expressed with its parameters implies that it is evaluated at the nominal trajectories, $\bar{\mathbf{x}}$ and $\bar{\mathbf{u}}$ (e.g. $\mathbf{f} \triangleq \mathbf{f}(\bar{\mathbf{x}}, \bar{\mathbf{u}})$).

We proceed by taking the quadratic expansion of $\mathcal{L}$ and $V(\mathbf{x}_{k+1}, t_{k+1})$ around the nominal trajectory. For $\mathcal{L}$, we obtain

$$\mathcal{L}(\mathbf{x}, \mathbf{u}) = \mathcal{L} + \ \delta\mathbf{x}^{\mathsf{T}}\mathcal{L}_{\mathbf{x}} + \ \delta\mathbf{u}^{\mathsf{T}}\mathcal{L}_{\mathbf{u}} + \frac{1}{2}\begin{bmatrix}\delta\mathbf{x}\\ \delta\mathbf{u}\end{bmatrix}^{\mathsf{T}}\begin{bmatrix}\mathcal{L}_{\mathbf{xx}} & \mathcal{L}_{\mathbf{xu}}\\ \mathcal{L}_{\mathbf{ux}} & \mathcal{L}_{\mathbf{uu}}\end{bmatrix}\begin{bmatrix}\delta\mathbf{x}\\ \delta\mathbf{u}\end{bmatrix} \ , \tag{2.15}$$

where we have omitted the time index subscript to simplify the notation. For V, we have

$$\begin{aligned}
V(\mathbf{x}_{k+1}, t_{k+1}) &= V^{(k+1)} + \ \delta\mathbf{x}_{k+1}^{\mathsf{T}} V_{\mathbf{x}}^{(k+1)} + \frac{1}{2}\, \delta\mathbf{x}_{k+1}^{\mathsf{T}} V_{\mathbf{xx}}^{(k+1)}\, \delta\mathbf{x}_{k+1} \\
&= V^{(k+1)} + \left(\mathbf{f}_{\mathbf{x}}\, \delta\mathbf{x}_k + \mathbf{f}_{\mathbf{u}}\, \delta\mathbf{u}_k\right)^{\mathsf{T}} V_{\mathbf{x}}^{(k+1)} + \frac{1}{2}\left(\mathbf{f}_{\mathbf{x}}\, \delta\mathbf{x}_k + \mathbf{f}_{\mathbf{u}}\, \delta\mathbf{u}_k\right)^{\mathsf{T}} V_{\mathbf{xx}}^{(k+1)} \left(\mathbf{f}_{\mathbf{x}}\, \delta\mathbf{x}_k + \mathbf{f}_{\mathbf{u}}\, \delta\mathbf{u}_k\right) \ ,
\end{aligned} \tag{2.16}$$

where the superscript of V and its gradient variants indicates the indexing of the discretized time, e.g. $V^{(k+1)} \triangleq V(\bar{\mathbf{x}}_{k+1}, t_{k+1})$, and the subscript indicates the vector that the gradient is taken with respect to, i.e. $V_{\mathbf{x}} \triangleq \nabla_{\mathbf{x}} V$ and $V_{\mathbf{xx}} \triangleq \nabla^2_{\mathbf{xx}} V$. Note that, as already explained, the absence of parameterization of a function (in this case V and its gradient variants) denotes that its evaluation is at the nominal trajectory, e.g. $V_{\mathbf{x}}^{(k)} \triangleq V_{\mathbf{x}}(\bar{\mathbf{x}}_k, t_k)$.

Next, we introduce the *action-value* function, $Q : \mathbb{R}^d \times \mathbb{R}^p \to \mathbb{R}$, which measures the accumulated cost by starting at a specified state and taking a particular action. Formally, it is defined as $Q(\mathbf{x}_k, \mathbf{u}_k) \triangleq \mathcal{L}(\mathbf{x}_k, \mathbf{u}_k) + V(\mathbf{x}_{k+1}, t_{k+1})$, which allows us to rewrite the value function as

$$V(\mathbf{x}, t) = \min_{\mathbf{u}} Q(\mathbf{x}, \mathbf{u}). \tag{2.17}$$

Because the two components of Q ($\mathcal{L}$ and $V^{(k+1)}$) are quadratic in $\delta\mathbf{x}$ and $\delta\mathbf{u}$ on their expanded form (Eqs. 2.15 and 2.16), defining Q allows us to work only with terms of the quadratic expansion of Q. Let us expand $Q(\mathbf{x}, \mathbf{u})$ around $\bar{\mathbf{x}}$ and $\bar{\mathbf{u}}$,

$$Q(\mathbf{x}, \mathbf{u}) = Q + Q_{\mathbf{x}}^{\mathsf{T}} \delta\mathbf{x} + Q_{\mathbf{u}}^{\mathsf{T}} \delta\mathbf{u} + \frac{1}{2} \begin{bmatrix} \delta\mathbf{x} \\ \delta\mathbf{u} \end{bmatrix}^{\mathsf{T}} \begin{bmatrix} Q_{\mathbf{xx}} & Q_{\mathbf{xu}} \\ Q_{\mathbf{ux}} & Q_{\mathbf{uu}} \end{bmatrix} \begin{bmatrix} \delta\mathbf{x} \\ \delta\mathbf{u} \end{bmatrix}, \tag{2.18}$$

and group terms in the expansions of $\mathcal{L}(\mathbf{x}, \mathbf{u})$ and $V(\mathbf{x}_{k+1}, t_{k+1})$ (in Eqs. 2.15 and 2.16) by the multiplicity of the orders of $\delta\mathbf{x}$ and $\delta\mathbf{u}$. We derive the groups:

$$Q = \mathcal{L} + V^{(k+1)} \tag{2.19}$$

$$Q_{\mathbf{x}} = \mathcal{L}_{\mathbf{x}} + \mathbf{f}_{\mathbf{x}}^{\mathsf{T}} V_{\mathbf{x}}^{(k+1)} \tag{2.20}$$

$$Q_{\mathbf{u}} = \mathcal{L}_{\mathbf{u}} + \mathbf{f}_{\mathbf{u}}^{\mathsf{T}} V_{\mathbf{x}}^{(k+1)} \tag{2.21}$$

$$Q_{\mathbf{xx}} = \mathcal{L}_{\mathbf{xx}} + \mathbf{f}_{\mathbf{x}}^{\mathsf{T}} V_{\mathbf{xx}}^{(k+1)} \mathbf{f}_{\mathbf{x}} \tag{2.22}$$

$$Q_{\mathbf{uu}} = \mathcal{L}_{\mathbf{uu}} + \mathbf{f}_{\mathbf{u}}^{\mathsf{T}} V_{\mathbf{xx}}^{(k+1)} \mathbf{f}_{\mathbf{u}} \tag{2.23}$$

$$Q_{\mathbf{xu}} = \mathcal{L}_{\mathbf{xu}} + \mathbf{f}_{\mathbf{x}}^{\mathsf{T}} V_{\mathbf{xx}}^{(k+1)} \mathbf{f}_{\mathbf{u}} \,. \tag{2.24}$$

Note that all Q's, $\mathcal{L}$'s and $\mathbf{f}$'s are evaluated at time t_k, while the V's are at the next time step, t_{k+1}. This forms the basis for DDP's approach to approximating the value function

and its gradients by working backwards in time, where the current Q's are computed from the V's in the next timestep. Because DDP runs backwards in time, the boundary condition for this dynamic programming approach is the terminal cost given by Φ in Eq. 2.2. So, $V_H = V(\bar{\mathbf{x}}_H, t_H) = \Phi(\bar{\mathbf{x}}_H)$ and analogously for V's gradient variants.

Now that we have expressions for V and Q, the next step is to find the optimal control $\mathbf{u}$ for each timestep. In Eq. 2.17, we have a simplified form of the value function, and so we must find a minimizer $\mathbf{u}^*$ to $Q(\mathbf{x}, \mathbf{u})$. Although the expansion of $Q(\mathbf{x}, \mathbf{u})$ at Eq. 2.18 does not contain any $\mathbf{u}$ terms, it can still be minimized over the controls $\mathbf{u}$, since $\delta\mathbf{u} = \mathbf{u} - \bar{\mathbf{u}}$ and $\bar{\mathbf{u}}$ is fixed. That is,

$$\min_{\mathbf{u}} Q(\mathbf{x}, \mathbf{u}) = \min_{\delta\mathbf{u}} Q(\mathbf{x},\, \delta\mathbf{u} + \bar{\mathbf{u}}) \,. \tag{2.25}$$

A control change minimizer $\delta\mathbf{u}^*$ to Eq. 2.17 is a stationary point of $Q(\mathbf{x}, \mathbf{u})$ w.r.t. $\delta\mathbf{u}$, provided that the optimization problem is unconstrained. Formally, $\nabla_{\delta\mathbf{u}} Q(\mathbf{x},\, \delta\mathbf{u}^* + \bar{\mathbf{u}}) = 0$. From Eq. 2.18, we derive the gradient

$$\nabla_{\delta\mathbf{u}} Q(\mathbf{x}, \mathbf{u}) = Q_{\mathbf{u}} + Q_{\mathbf{ux}}\, \delta\mathbf{x} + Q_{\mathbf{uu}}\, \delta\mathbf{u} \,, \tag{2.26}$$

and setting it to zero, we solve for the optimal control change

$$\delta\mathbf{u}^* = -Q_{\mathbf{uu}}^{-1}(Q_{\mathbf{u}} + Q_{\mathbf{ux}}\, \delta\mathbf{x}) \,. \tag{2.27}$$

Note that this control change is addressing only a particular time t_k. In order to compute $\delta\mathbf{u}^*_{k-1}$, we must find the Q's for time t_{k-1} using Eqs. 2.19 - 2.24. This requires the evaluation of $V^{(k)}$, $V^{(k)}_{\mathbf{x}}$, and $V^{(k)}_{\mathbf{xx}}$. To obtain these, we use the newly found $\delta\mathbf{u}^*$ at Eq. 2.27 to evaluate

$V(\mathbf{x}, t)$ (Eq. 2.17):

$$
\begin{aligned}
V(\mathbf{x}, t) &= Q(\mathbf{x}, \mathbf{u}^*) \\
&= Q + Q_{\mathbf{x}}^{\mathsf{T}}\,\delta\mathbf{x} + Q_{\mathbf{u}}^{\mathsf{T}}\,\delta\mathbf{u}^* + \frac{1}{2}\begin{bmatrix}\delta\mathbf{x}\\ \delta\mathbf{u}^*\end{bmatrix}^{\mathsf{T}}\begin{bmatrix}Q_{\mathbf{xx}} & Q_{\mathbf{xu}}\\ Q_{\mathbf{ux}} & Q_{\mathbf{uu}}\end{bmatrix}\begin{bmatrix}\delta\mathbf{x}\\ \delta\mathbf{u}^*\end{bmatrix} \\
&= Q + Q_{\mathbf{x}}^{\mathsf{T}}\,\delta\mathbf{x} - Q_{\mathbf{u}}^{\mathsf{T}}Q_{\mathbf{uu}}^{-1}(Q_{\mathbf{u}} + Q_{\mathbf{ux}}\,\delta\mathbf{x}) + \frac{1}{2}\,\delta\mathbf{x}^{\mathsf{T}}Q_{\mathbf{xx}}\,\delta\mathbf{x} \\
&\quad + \frac{1}{2}(Q_{\mathbf{u}}^{\mathsf{T}} + \delta\mathbf{x}^{\mathsf{T}}Q_{\mathbf{xu}})Q_{\mathbf{uu}}^{-1}Q_{\mathbf{uu}}Q_{\mathbf{uu}}^{-1}(Q_{\mathbf{u}} + Q_{\mathbf{ux}}\,\delta\mathbf{x}) && (2.28)\\
&\quad - \delta\mathbf{x}^{\mathsf{T}}Q_{\mathbf{xu}}Q_{\mathbf{uu}}^{-1}(Q_{\mathbf{u}} + Q_{\mathbf{ux}}\,\delta\mathbf{x}) && (2.29)\\
&= \left[Q - Q_{\mathbf{u}}^{\mathsf{T}}Q_{\mathbf{uu}}^{-1}Q_{\mathbf{u}} + \frac{1}{2}Q_{\mathbf{u}}^{\mathsf{T}}Q_{\mathbf{uu}}^{-1}Q_{\mathbf{u}}\right] && (2.30)\\
&\quad + \delta\mathbf{x}^{\mathsf{T}}\left[Q_{\mathbf{x}} - Q_{\mathbf{xu}}Q_{\mathbf{uu}}^{-1}Q_{\mathbf{u}} + Q_{\mathbf{xu}}Q_{\mathbf{uu}}^{-1}Q_{\mathbf{u}} - Q_{\mathbf{xu}}Q_{\mathbf{uu}}^{-1}Q_{\mathbf{u}}\right] && (2.31)\\
&\quad + \frac{1}{2}\,\delta\mathbf{x}^{\mathsf{T}}\left[Q_{\mathbf{xx}} + Q_{\mathbf{xu}}Q_{\mathbf{uu}}^{-1}Q_{\mathbf{ux}} - 2Q_{\mathbf{xu}}Q_{\mathbf{uu}}^{-1}Q_{\mathbf{ux}}\right]\,\delta\mathbf{x} && (2.32)\\
&= \left[Q - \frac{1}{2}Q_{\mathbf{u}}^{\mathsf{T}}Q_{\mathbf{uu}}^{-1}Q_{\mathbf{u}}\right] + \delta\mathbf{x}^{\mathsf{T}}\left[Q_{\mathbf{x}} - Q_{\mathbf{xu}}Q_{\mathbf{uu}}^{-1}Q_{\mathbf{u}}\right] + \frac{1}{2}\,\delta\mathbf{x}^{\mathsf{T}}\left[Q_{\mathbf{xx}} - Q_{\mathbf{xu}}Q_{\mathbf{uu}}^{-1}Q_{\mathbf{ux}}\right]\,\delta\mathbf{x} \\
& && (2.33)
\end{aligned}
$$

Note that Expression 2.33 is of second order in $\delta\mathbf{x}$. As a result, we can also expand $V(\mathbf{x}, t)$ quadratically around $\bar{\mathbf{x}}$,

$$V(\mathbf{x}, t) = V + \delta\mathbf{x}^{\mathsf{T}}V_{\mathbf{x}} + \frac{1}{2}\,\delta\mathbf{x}^{\mathsf{T}}V_{\mathbf{xx}}\,\delta\mathbf{x} \quad , \tag{2.34}$$

and match the terms based on their order in $\delta\mathbf{x}$. Finally, we obtain

$$V = Q - \frac{1}{2}Q_{\mathbf{u}}^{\mathsf{T}}Q_{\mathbf{uu}}^{-1}Q_{\mathbf{u}} \tag{2.35}$$

$$V_{\mathbf{x}} = Q_{\mathbf{x}} - Q_{\mathbf{xu}}Q_{\mathbf{uu}}^{-1}Q_{\mathbf{u}} \tag{2.36}$$

$$V_{\mathbf{xx}} = Q_{\mathbf{xx}} - Q_{\mathbf{xu}}Q_{\mathbf{uu}}^{-1}Q_{\mathbf{ux}} \tag{2.37}$$

Assuming these are addressing time t_k, we can now compute the Q's for time t_{k-1} (using Eqs. 2.19 - 2.24) and proceedingly, find $\delta\mathbf{u}_{k-1}^*$ with Eq. 2.27. We proceed with this iterative backward scheme to find a control change for every time step within the time horizon. These

control changes will be used to create a new nominal control sequence, $\bar{\mathbf{u}} \leftarrow \bar{\mathbf{u}} + \delta\mathbf{u}^*$, which will generate a new nominal state trajectory. The complete DDP algorithm is described in Alg. 1.

Algorithm 1: Differential Dynamic Programming

Input : Initial state $\mathbf{x}_0$

Time horizon H

Discrete dynamics $f(\mathbf{x}, \mathbf{u})\delta t$

Cost function: Lagrangian $\mathcal{L}$ and terminal cost function Φ

Output: Locally optimal control sequence $(\mathbf{u}_0, \ldots, \mathbf{u}_{H-1})$

Initialize nominal control sequence $(\mathbf{u}_0, \ldots, \mathbf{u}_{H-1})$

Initialize zero control change $\delta\mathbf{u}_k(\cdot) = 0$ for $k = 0, \ldots, H-1$

while *cost has not converged* **do**

  for $k = 0, \ldots, H-1$ **do**

    Update nominal control: $\mathbf{u}_k \leftarrow \mathbf{u}_k + \delta\mathbf{u}_k(\,\delta\mathbf{x})$, where $\delta\mathbf{x} = \mathbf{x}_k - \bar{\mathbf{x}}_k$

    Update nominal state: $\bar{\mathbf{x}}_k \leftarrow \mathbf{x}_k$

    Compute $\mathbf{x}_{k+1}$ using discrete dynamics as in Eq. 2.5: $\mathbf{x}_{k+1} \leftarrow \bar{\mathbf{x}}_k + f(\bar{\mathbf{x}}_k, \mathbf{u}_k)\delta t$

    Compute running cost and its gradients evaluated at $\bar{\mathbf{x}}_k$ and $\mathbf{u}_k$:

      $\mathcal{L}^{(k)}$, $\mathcal{L}_{\mathbf{u}}^{(k)}$, $\mathcal{L}_{\mathbf{x}}^{(k)}$, $\mathcal{L}_{\mathbf{xx}}^{(k)}$, $\mathcal{L}_{\mathbf{uu}}^{(k)}$, $\mathcal{L}_{\mathbf{xu}}^{(k)}$

    Compute dynamics gradients $\mathbf{f}_{\mathbf{x}}^{(k)}$ and $\mathbf{f}_{\mathbf{u}}^{(k)}$ as in Eq. 2.5

  $V^{(H)} \leftarrow \Phi(\mathbf{x}_H)$

  $V_{\mathbf{x}}^{(H)} \leftarrow \Phi_{\mathbf{x}}(\mathbf{x}_H)$

  $V_{\mathbf{xx}}^{(H)} \leftarrow \Phi_{\mathbf{xx}}(\mathbf{x}_H)$

  for $k = H-1, \ldots, 0$ **do**

    Compute $Q^{(k)}$, $Q_{\mathbf{x}}^{(k)}$, $Q_{\mathbf{u}}^{(k)}$, $Q_{\mathbf{xx}}^{(k)}$, $Q_{\mathbf{uu}}^{(k)}$, $Q_{\mathbf{xu}}^{(k)}$ using Eqs. 2.19 – 2.24

    Compute $V^{(k)}$, $V_{\mathbf{x}}^{(k)}$, $V_{\mathbf{xx}}^{(k)}$ using Eqs. 2.35 – 2.37

    Compute $\delta\mathbf{u}_k(\cdot)$ using Eq. 2.27

return $(\mathbf{u}_0, \ldots, \mathbf{u}_{H-1})$

2.2 Model Learning with Gaussian Processes

In this section, we introduce the Gaussian processes (GPs) as a Bayesian linear regression method and we motivate the use of GPs to model system dynamics.

2.2.1 Bayesian Linear Regression

Supervised learning, another of the three branches of machine learning, is concerned with extracting information and learning patterns from a dataset of input-output pairs. More precisely, the objective is to learn a function that approximates the the true function that generated the dataset provided; thus allowing us to predict the output corresponding to an unseen input data.

As mentioned in Chapter 1, we consider a problem in which we do not know the form of the dynamics. As a result, supervised learning provides a useful framework to model the system dynamics and be able to predict future states. First, we introduce Bayesian linear regression as a supervised learning method to approximate the dynamics function.

We approximate the true function that generated a dataset $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ with a linear function with the output corrupted by Gaussian noise

$$f(\mathbf{x}) = \mathbf{x}^\mathsf{T}\mathbf{w} \ , \qquad y = f(\mathbf{x}) + \epsilon \tag{2.38}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the input, $\mathbf{w} \in \mathbb{R}^n$ are the model parameters, $y \in \mathbb{R}$ is the scalar output, and the noise is an i.i.d. Gaussian distribution with zero mean and variance σ^2, $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$. Defining our model as such, we can obtain the *likelihood*, defined as the probability density of the observations $\{y_i\}$ given the input data $\{\mathbf{x}_i\}$ and the parameters $\mathbf{w}$:

$$p(\mathbf{y} \mid \mathbf{X}, \mathbf{w}) = \prod_{i=1}^{N} p(y_i \mid \mathbf{x}_i, \mathbf{w}) \tag{2.39}$$

$$= \prod_{i=1}^{N} \mathcal{N}(y_i \mid \mathbf{x}_i^\mathsf{T}\mathbf{w}, \sigma_\epsilon^2) \tag{2.40}$$

$$= \mathcal{N}(\mathbf{y} \mid \mathbf{X}\mathbf{w}, \sigma^2\mathbf{I}) \ , \tag{2.41}$$

where $\mathbf{X} = [\mathbf{x}_1 \cdots \mathbf{x}_N]^\mathsf{T}$ and $\mathbf{y} = [y_1 \cdots y_N]^\mathsf{T}$. If we specify a *prior* distribution over the parameters $\mathbf{w}$, we can derive an expression for the *posterior* distribution over the weights, given our dataset $(\mathbf{X}, \mathbf{y})$. Thus, assuming the prior $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \Sigma_p)$, we obtain the posterior

$$p(\mathbf{w} \mid \mathbf{y}, \mathbf{X}) = \frac{p(\mathbf{y} \mid \mathbf{X}, \mathbf{w})p(\mathbf{w})}{p(\mathbf{y} \mid \mathbf{X})} \tag{2.42}$$

$$= \mathcal{N}(\mathbf{w} \mid \frac{1}{\sigma_\epsilon^2} A^{-1}\mathbf{X}\mathbf{y}, A^{-1}) \,, \tag{2.43}$$

where $A = \sigma_\epsilon^{-2}\mathbf{X}\mathbf{X}^\mathsf{T} + \Sigma_p^{-1}$. Because the posterior (Eq. 2.43) has a Gaussian form, the mean is also the *maximum a posterior* (MAP) of the random variable $\mathbf{w}$.

To predict a new test input $\mathbf{x}_*$, we average the linear predictions over all possible weights $\mathbf{w}$, weighted by their posterior probability. Defining $f_* \triangleq f(\mathbf{x}_*)$, we obtain a Gaussian distribution over our prediction

$$p(f_* \mid \mathbf{x}_*, \mathbf{X}, \mathbf{y}) = \int p(f_* \mid \mathbf{x}_*, \mathbf{w})p(\mathbf{w} \mid \mathbf{X}, \mathbf{y}) \, \mathrm{d}\mathbf{w} \tag{2.44}$$

$$= \mathcal{N}(f_* \mid \frac{1}{\sigma_\epsilon^2}\mathbf{x}_*^\mathsf{T} A^{-1}\mathbf{X}\mathbf{y}, \mathbf{x}_*^\mathsf{T} A^{-1}\mathbf{x}_*) \,. \tag{2.45}$$

We defer the derivation of the expressions in this section to Rasmussen and Williams [2005].

Bayesian linear regression is a supervised learning method grounded on the analysis of probability distributions over the parameter weights $\mathbf{w}$. In the next section, we extend this method from weights-space to function-space, in which we now analyze probability distributions over function spaces.

2.2.2 Gaussian Processes

We can generalize the Bayesian linear regression method detailed in the previous section with *Gaussian processes* (GPs). A GP is a distribution over functions and it is formally defined as follows

Definition 2.1 *A **Gaussian process** is a collection of random variables, any finite number of which have a joint Gaussian distribution [Rasmussen and Williams, 2005, Deisenroth, 2010].*

A GP is fully defined by its mean function and covariance, so that

$$f(\cdot) \sim \mathcal{GP}\left(m(\cdot), k(\cdot,\cdot)\right) \ , \tag{2.46}$$

where

$$m(\mathbf{x}) = \underset{f}{\mathbb{E}}[f(\mathbf{x})] \qquad \text{and} \qquad k(\mathbf{x},\mathbf{x}') = \underset{f}{\mathbb{C}}[f(\mathbf{x}), f(\mathbf{x}')] \ . \tag{2.47}$$

The covariance function $k(\mathbf{x},\mathbf{x}')$ is also called *kernel*. Usually, the index set of a stochastic process is time, i.e. that at each timestep, we observe the realization of a random variable. In the case for the GP, however, the inputs $\mathbf{x}$ are the indices of the stochastic process, and the random variable is the value of $f(\cdot)$ at a particular index $\mathbf{x}$, i.e. simply $f(\mathbf{x})$.

For the purpose of introduction, we will only consider zero-mean function. If that's the case, when we have a dataset of training inputs $\{\mathbf{x}_i\}_{i=1}^N$ and a test input $\mathbf{x}_* \in \mathbb{R}^n$, the output values $\mathbf{y}$, with $\mathbf{y}_i \triangleq f(\mathbf{x}_i) + \epsilon$, and $\mathbf{f}_* \triangleq f(\mathbf{x}_*)$ are described by a finite multivariate Gaussian distribution, given by

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K} + \sigma_\epsilon^2 \mathbf{I} & \mathbf{k}_* \\ \mathbf{k}_*^\mathsf{T} & k_{**} \end{bmatrix}\right) \ , \tag{2.48}$$

where $\mathbf{K}$ is called the Gram matrix and $\mathbf{K}_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$, $\mathbf{k}_{*i} = k(\mathbf{x}_*, \mathbf{x}_i)$, and $k_{**} = k(\mathbf{x}_*, \mathbf{x}_*)$. Note that we have the training output values $\mathbf{y}$; therefore, we must find the posterior probability distribution of $\mathbf{f}_*$. It turns out it is, in fact, Gaussian and described as

$$\mathbf{f}_* \sim \mathcal{N}\left(\mathbf{k}_*^\mathsf{T}(\mathbf{K} + \sigma_\epsilon^2\mathbf{I})^{-1}\mathbf{y}, k_{**} - \mathbf{k}_*^\mathsf{T}(\mathbf{K} + \sigma_\epsilon^2\mathbf{I})^{-1}\mathbf{k}_*\right) \ . \tag{2.49}$$

We defer the derivation of the expressions above to Rasmussen and Williams [2005]. This concludes the introduction to GPs, as we have fully defined how to do inference of a new test input $\mathbf{x}_*$, given a input-output dataset $(\mathbf{X}, \mathbf{y})$.

2.3 Probabilistic Differential Dynamic Programming

Thus far, we have introduced DDP (an algorithm that solves the optimal control problem by computing local approximations of the value function and its gradients) and GPs (a Bayesian non-parametric method for regression). In this section, we introduce the algorithm of which this thesis is founded upon, Probabilistic Differential Dynamic Programming (PDDP), developed by Pan and Theodorou [2014]. In this model-based RL algorithm, the dynamics is learned with a GP and a control sequence is optimized by considering a belief state at each time-step.

First, we consider discrete-time dynamics given by

$$\mathbf{x}_{k+1} = \mathbf{f}^*(\mathbf{x}_k, \mathbf{u}_k) \ , \tag{2.50}$$

and we aim to learn this dynamics function $\mathbf{f}^*$ with the GP method we described in the previous section. To learn a dynamics model with a GP, we first assume we have a dataset of state-control-transition tuples $\{(\mathbf{x}_i, \mathbf{u}_i, \boldsymbol{\delta}_i)\}_{i=1}^N$, where the transition is the change in state by applying a control, i.e. $\boldsymbol{\delta}_k \triangleq \mathbf{x}_{k+1} - \mathbf{x}_k$. We can use a GP to model these state transitions by using $\tilde{\mathbf{x}} \triangleq [\mathbf{x}^\mathsf{T} \ \mathbf{u}^\mathsf{T}]^\mathsf{T} \in \mathbb{R}^{d+p}$ as the inputs, and the transitions as the outputs. In the previous section, we describe how to use GPs for inference of a real-valued output. In our case, however, we must predict multivariate outputs. We address this by having a separate GP for each output dimension, thus assuming that there is not correlation between the outputs. As a result, we will approximate the dynamics model $\mathbf{f}^*$ with a bayesian model $\mathbf{f} \triangleq [\mathbf{f}_1 \ \cdots \ \mathbf{f}_d]^\mathsf{T}$, where for each output dimension each $a = 1, \ldots, d$, $\mathbf{f}_a$ is an independent GP with zero-mean function and an squared exponential kernel given by

$$k_a(\tilde{\mathbf{x}}, \tilde{\mathbf{x}}') = \alpha_a^2 \exp\left(-\frac{1}{2}(\tilde{\mathbf{x}} - \tilde{\mathbf{x}}')^\mathsf{T} \boldsymbol{\Lambda}_a^{-1} (\tilde{\mathbf{x}} - \tilde{\mathbf{x}}')\right) \ , \tag{2.51}$$

where α_a^2 is the variance of $\mathbf{f}_a$ and $\boldsymbol{\Lambda}_a \triangleq \text{diag}\left([\ell_{a1}^2, \ldots, \ell_{ad}^2]\right)$ is defined by the characteristic

lengthscales ℓ_{ai} for $i = 1, \ldots, (d+p)$. Then, for a test input $\tilde{\mathbf{x}}_*$

$$m_{\mathbf{f}_a}(\tilde{\mathbf{x}}_*) = \mathop{\mathbb{E}}_{\mathbf{f}_a}[\boldsymbol{\delta}_*] = \mathbf{k}_*^\mathsf{T}(\mathbf{K} + \sigma_\epsilon^2\mathbf{I})^{-1}\mathbf{y} = \mathbf{k}_*^\mathsf{T}\boldsymbol{\beta} \tag{2.52}$$

$$\sigma_{\mathbf{f}_a}^2(\tilde{\mathbf{x}}_*) = \mathop{\mathbb{V}}_{\mathbf{f}_a}[\boldsymbol{\delta}_*] = k_{**} - \mathbf{k}_*^\mathsf{T}(\mathbf{K} + \sigma_\epsilon^2\mathbf{I})^{-1}\mathbf{k}_* \,, \tag{2.53}$$

where $\boldsymbol{\beta} \triangleq (\mathbf{K} + \sigma_\epsilon^2\mathbf{I})^{-1}\mathbf{y}$.

To extend DDP to a belief space setting, our new augmented state will now have to be a belief. In the case of PDDP, the belief is represented by a Gaussian distribution. Given a Gaussian belief $p(\tilde{\mathbf{x}}) = \mathcal{N}(\tilde{\mathbf{x}} \mid \tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\Sigma}})$, we define an augmented state representing this belief as

$$\mathbf{z} \triangleq \begin{bmatrix} \tilde{\boldsymbol{\mu}} \\ \mathrm{vec}(\tilde{\boldsymbol{\Sigma}}) \end{bmatrix} \,, \tag{2.54}$$

where $\mathrm{vec}(\tilde{\boldsymbol{\Sigma}})$ is the vectorized form of the matrix. Given a Gaussian belief at time-step k, we would like to predict the belief at the next time-step $k+1$ with our GP model. The next section details this step.

2.3.1 Gaussian Belief One-Step Propagation via Moment Matching

We start by assuming a Gaussian belief $p(\mathbf{x}_k) = \mathcal{N}(\mathbf{x}_k \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$, which gives rise to the joint state-control Gaussian distribution $p(\tilde{\mathbf{x}}_k) = \mathcal{N}(\tilde{\mathbf{x}}_k \mid \tilde{\boldsymbol{\mu}}_k, \tilde{\boldsymbol{\Sigma}}_k)$. The objective is to propagate this state-control input distribution to the next time-step and find $p(\mathbf{x}_{k+1})$. We do this by first describing the probability distribution of the transition, given by

$$p(\boldsymbol{\delta}_k) = \int p(\mathbf{f}(\tilde{\mathbf{x}}_k) \mid \tilde{\mathbf{x}}_k) p(\tilde{\mathbf{x}}_k) \, \mathrm{d}\tilde{\mathbf{x}}_k \,. \tag{2.55}$$

The above is not only analytically intractable to compute, but also non-Gaussian due to the non-linearity of the GP described by $\mathbf{f}$. Instead, we approximate $p(\tilde{\mathbf{x}}_{k+1})$ with a Gaussian by analytically matching the first and second moments of the two distributions [Deisenroth et al., 2015, Deisenroth, 2010]. Thus, the belief of the next time-step is given by $p(\mathbf{x}_{k+1}) =$

$\mathcal{N}(\mathbf{x}_{k+1} \mid \boldsymbol{\mu}_{k+1}, \boldsymbol{\Sigma}_{k+1})$ with

$$\boldsymbol{\mu}_{k+1} = \boldsymbol{\mu}_k + \boldsymbol{\mu}_\delta \tag{2.56}$$

$$\boldsymbol{\Sigma}_{k+1} = \boldsymbol{\Sigma}_k + \boldsymbol{\Sigma}_\delta + \mathbb{C}[\mathbf{x}_k, \boldsymbol{\delta}_k] + \mathbb{C}[\boldsymbol{\delta}_k, \mathbf{x}_k] \tag{2.57}$$

where $\boldsymbol{\mu}_\delta \triangleq \mathbb{E}[\boldsymbol{\delta}_k] \in \mathbb{R}^d$ and $\boldsymbol{\Sigma}_\delta \triangleq \mathbb{V}[\boldsymbol{\delta}_k] \in \mathbb{R}^{d \times d}$. Note that these expectations and covariances are taken with respect to the randomness of both the GP, $\mathbf{f}$, and the input distribution, $p(\tilde{\mathbf{x}}_k)$.

We begin by computing $\boldsymbol{\mu}_\delta$ for each output dimension $a = 1, \ldots, d$. Using the law of iterated expectations, we get

$$\boldsymbol{\mu}_\delta^a = \mathop{\mathbb{E}}_{\tilde{\mathbf{x}}}\left[\mathop{\mathbb{E}}_{\mathbf{f}_a}[\mathbf{f}_a(\tilde{\mathbf{x}}) \mid \tilde{\mathbf{x}}]\right] = \mathop{\mathbb{E}}_{\tilde{\mathbf{x}}}[m_{\mathbf{f}_a}(\tilde{\mathbf{x}})] \tag{2.58}$$

$$= \int m_{\mathbf{f}_a}(\tilde{\mathbf{x}}) \mathcal{N}(\tilde{\mathbf{x}} \mid \tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\Sigma}}) \, \mathrm{d}\tilde{\mathbf{x}} \tag{2.59}$$

$$= (\boldsymbol{\beta}^a)^\mathsf{T} \mathbf{q}^a \,, \tag{2.60}$$

where $\mathbf{q}^a \in \mathbb{R}^N$ and

$$\mathbf{q}_i^a = \int k_a(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}) \mathcal{N}(\tilde{\mathbf{x}} \mid \tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\Sigma}}) \, \mathrm{d}\tilde{\mathbf{x}} \tag{2.61}$$

$$= \frac{\alpha_a^2}{|\tilde{\boldsymbol{\Sigma}} \boldsymbol{\Lambda}_a^{-1} + \mathbf{I}|^2} \exp\left(-\frac{1}{2} \boldsymbol{\nu}_i^\mathsf{T} (\tilde{\boldsymbol{\Sigma}} + \boldsymbol{\Lambda}_a)^{-1} \boldsymbol{\nu}_i\right) \,, \tag{2.62}$$

where $\boldsymbol{\nu}_i \triangleq \tilde{\mathbf{x}}_i - \tilde{\boldsymbol{\mu}}_k$. For the covariance $\boldsymbol{\Sigma}_\delta$, we use the law of total variance and we get that for each output pair dimension $a, b = 1, \ldots, d$

$$[\boldsymbol{\Sigma}_\delta]_{ab} = \mathop{\mathbb{E}}_{\mathbf{f}, \tilde{\mathbf{x}}}[\boldsymbol{\delta}_a \boldsymbol{\delta}_b] - \boldsymbol{\mu}_\delta^a \boldsymbol{\mu}_\delta^b + \delta_{ab} \mathop{\mathbb{E}}_{\tilde{\mathbf{x}}}\left[\mathop{\mathbb{V}}_{\mathbf{f}}[\boldsymbol{\delta}_a \mid \tilde{\mathbf{x}}]\right] \,, \tag{2.63}$$

where the unbolded delta δ_{ab} is the kronecker delta function. We can derive that

$$\mathop{\mathbb{E}}_{\mathbf{f},\tilde{\mathbf{x}}}[\boldsymbol{\delta}_a \boldsymbol{\delta}_b] = \boldsymbol{\beta}_a^\mathsf{T} \boldsymbol{\Phi} \boldsymbol{\beta}_b \;, \qquad \text{where} \tag{2.64}$$

$$\boldsymbol{\Phi} \triangleq \int k_a(\tilde{\mathbf{x}}, \tilde{\mathbf{X}})^\mathsf{T} k_b(\tilde{\mathbf{x}}, \tilde{\mathbf{X}}) p(\tilde{\mathbf{x}}) \, \mathrm{d}\tilde{\mathbf{x}} \tag{2.65}$$

$$\boldsymbol{\Phi}_{ij} \triangleq \frac{k_a(\tilde{\mathbf{x}}_i, \tilde{\boldsymbol{\mu}}) k_b(\tilde{\mathbf{x}}_j, \tilde{\boldsymbol{\mu}})}{|\tilde{\boldsymbol{\Sigma}}(\boldsymbol{\Lambda}_a^{-1} + \boldsymbol{\Lambda}_b^{-1}) + \mathbf{I}|^2} \exp\left(\frac{1}{2} \mathbf{v}_{ij}^\mathsf{T} \mathbf{T}^{-1} \mathbf{v}_{ij}\right) \;, \tag{2.66}$$

where

$$\mathbf{T} \triangleq \boldsymbol{\Lambda}_a^{-1} + \boldsymbol{\Lambda}_b^{-1} + \tilde{\boldsymbol{\Sigma}}^{-1} \;, \qquad \mathbf{v}_{ij} \triangleq \boldsymbol{\Lambda}_a^{-1} \boldsymbol{\nu}_i + \boldsymbol{\Lambda}_b^{-1} \boldsymbol{\nu}_j \;. \tag{2.67}$$

The additional diagonal terms are

$$\mathop{\mathbb{E}}_{\tilde{\mathbf{x}}}\left[\mathop{\mathbb{V}}_{\mathbf{f}}[\boldsymbol{\delta}_a \mid \tilde{\mathbf{x}}]\right] = \alpha_a^2 - \mathbf{Tr}\left((\mathbf{K}_a + \sigma_\epsilon^2 \mathbf{I})^{-1} \boldsymbol{\Phi}\right) \;. \tag{2.68}$$

Additionally, we obtain the following covariance terms

$$\mathbb{C}[\mathbf{x}_k, \boldsymbol{\delta}_k^a] = \sum_{i=1}^{N} \boldsymbol{\beta}_i^a \mathbf{q}_i^a \tilde{\boldsymbol{\Sigma}}_k (\tilde{\boldsymbol{\Sigma}}_k + \boldsymbol{\Lambda}_a)^{-1} \boldsymbol{\nu}_i \tag{2.69}$$

All the derivations for the above expressions are deferred to Deisenroth and Rasmussen [2011], Deisenroth et al. [2015], Deisenroth [2010], Pan and Theodorou [2014].

With this we can define the dynamics in terms of the belief z

$$\mathbf{z}_{k+1} = \mathbf{f}(\mathbf{z}_k, \mathbf{u}_k) \;. \tag{2.70}$$

We can find the Jacobian matrices of this belief space dynamics function, $\mathbf{f_x}$ and $\mathbf{f_u}$ as

$$\mathbf{f_x} = \nabla_\mathbf{x} \mathbf{f} = \begin{bmatrix} \nabla_{\boldsymbol{\mu}_k} \boldsymbol{\mu}_{k+1} & \nabla_{\boldsymbol{\Sigma}_k} \boldsymbol{\mu}_{k+1} \\ \nabla_{\boldsymbol{\mu}_k} \boldsymbol{\Sigma}_{k+1} & \nabla_{\boldsymbol{\Sigma}_k} \boldsymbol{\Sigma}_{k+1} \end{bmatrix} \tag{2.71}$$

$$\mathbf{f_u} = \nabla_\mathbf{u} \mathbf{f} = \begin{bmatrix} \nabla_{\mathbf{u}_k} \boldsymbol{\mu}_{k+1} \\ \nabla_{\mathbf{u}_k} \boldsymbol{\Sigma}_{k+1} \end{bmatrix} \;, \tag{2.72}$$

in which we also defer the derivations to Pan and Theodorou [2014].

2.3.2 Belief cost

The cost of the belief is incurred by taking an expectation of the state-control cost with respect to the belief. Thus, if the state-control cost is defined as

$$\mathcal{L}(\mathbf{x}, \mathbf{u}) = (\mathbf{x} - \mathbf{x}_g)^\mathsf{T}\mathbf{Q}(\mathbf{x} - \mathbf{x}_g) + \mathbf{u}^\mathsf{T}\mathbf{R}\mathbf{u} \,, \tag{2.73}$$

where $\mathbf{x}_g$ is the target goal state. Then the cost $\mathcal{L}^\mathbf{z}$ for a belief $p(\mathbf{x}) = \mathcal{N}(\mathbf{x} \mid \mu, \Sigma)$ is given as

$$\mathcal{L}^\mathbf{z}(\mathbf{z}, \mathbf{u}) = \mathop{\mathbb{E}}_{\mathbf{x}}[\mathcal{L}(\mathbf{x}, \mathbf{u})] = \mathbf{Tr}(\mathbf{Q}\boldsymbol{\Sigma}) + (\boldsymbol{\mu} - \mathbf{x}_g)^\mathsf{T}\mathbf{Q}(\boldsymbol{\mu} - \mathbf{x}_g) + \mathbf{u}^\mathsf{T}\mathbf{R}\mathbf{u} \,. \tag{2.74}$$

The gradients of the cost with respect to the belief and the control are easily obtained and deferred to Pan and Theodorou [2014].

Having defined the dynamics function $\mathbf{f}$, the cost function $\mathcal{L}^\mathbf{z}$, and their respective gradients in belief space, we can simply apply DDP.

Chapter 3

Methods

In this thesis, we address the problem that PDDP suffers due to the approximation error of using a Gaussian distribution to represent the state belief. This problem arises from the fact that the dynamics, modeled by a GP, can be highly non-linear, and thus the Gaussian belief, when propagated to the next timestep, can be highly non-Gaussian.

To address this, we use a method developed by Hardy et al. [2015], Havlak and Campbell [2013] to split a Gaussian distribution into a Gaussian mixture and then propagate each of the Gaussians into the next timestep by using the analytical moment matching method we have discussed.

3.1 Gaussian Mixture Splitting

First, we address how to split a Gaussian distribution into a mixture of Gaussians. We use the approach developed by Hardy et al. [2015], Havlak and Campbell [2013], wherein we describe a Gaussian distribution by a mixture of equally spaced Gaussians along the direction of (the eigenvector corresponding to) the greatest eigenvalue of the covariance matrix. There is no exact analytical method to describe a Gaussian with a mixture of Gaussians with smaller covariances. However, we can approximate the original Gaussian by solving an optimization

problem. Formally, if we are to split $p(\mathbf{x}) = \mathcal{N}(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$ into a mixture of n Gaussians, then we must solve

$$\min_{\Delta, w_i, \boldsymbol{\mu}_i} D \left[\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}) \parallel \sum_{i=1}^{n} w^i \cdot \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \right] \text{, subject to} \tag{3.1}$$

$$\boldsymbol{\mu}_{i+1} - \boldsymbol{\mu}_i = \Delta \mathbf{v}^* \tag{3.2}$$

$$\sum_{i=1}^{n} w^i = 1 \,, \tag{3.3}$$

where $\Delta \in \mathbb{R}$, D is some distance metric between two probability distributions, and $\mathbf{v}^*$ is the direction along which we aim to split the original Gaussian. One probability distance metric D that allows us to compute the distance between the original Gaussian and a mixture of Gaussian analytically is the integral squared difference (ISD), defined as

$$D[p \parallel q] = \int (p(x) - q(x))^2 \, \mathrm{d}x \,. \tag{3.4}$$

We can simplify the problem even more by consdering approximating the standard normal distribution with a Gaussian mixture. In this manner, we only need to solve this "standard" optimization problem once offline, and we simply apply a rotation and translation transformation to the Gaussian mixture solution in order to approximate a non-standard Gaussian. We defer mathematical details of this method to Havlak and Campbell [2013].

3.1.1 Criterion for Splitting

The approximation error arises from the nonlinearity of the GP local to the input Gaussian distribution, which can cause the true posterior distribution to be significantly non-Gaussian. Splitting the original Gaussian into a mixture of Gaussians with smaller covariances allows the posterior mixture approximation to have lower error. This is due to the observation that the GP is more linear when considering smaller neighborhoods in the input space, which would be the case for each of the new Gaussians with smaller covariances.

The criterion for splitting a Gaussian, therefore, will be to examine whether the true belief, after

propagating a Gaussian input distribution through the GP, is sufficiently non-Gaussian. We do this by using a distance metric G from a Gaussian distribution and we define the criterion as a threshold distance. This non-Gaussianity metric G here will simply be the difference in the kurtosis of a Gaussian and the output distribution. The kurtosis of a random variable X is defined as:

$$K[X] \triangleq \frac{\mathbb{E}[(X-\mu)^4]}{\mathbb{E}[(X-\mu)^2]^2} \,, \tag{3.5}$$

where $\mu \triangleq \mathbb{E}[X]$. Thus, the non-Gaussianity metric of a distribution q is

$$G[q] = \big|K[q] - K[\mathcal{N}]\big| \,. \tag{3.6}$$

In our case, the distribution q will be the true belief at the next time-step after propagating a Gaussian belief through the GP. However, it is analytically intractable to compute this distribution. What we do instead, is sample points from the original Gaussian belief, propagate them through the GP dynamics, and compute an estimate of the kurtosis of the output belief, given by

$$\hat{K}\left[\{\mathbf{x}_i\}_{i=1}^N\right] \triangleq \frac{\frac{1}{N}\sum_{i=1}^N (\mathbf{x}-\boldsymbol{\mu})^4}{\left(\frac{1}{N}\sum_{i=1}^N (\mathbf{x}-\boldsymbol{\mu})^2\right)^2} \,, \tag{3.7}$$

where $\{\mathbf{x}_i\}_{i=1}^N$ is the set of N samples after being propagated through the GP. Finally, we pick a specific threshold, ϵ_K, and if $G\left[\{\mathbf{x}_i\}\right] > \epsilon_K$, then we split the Gaussian and propagate each of them independently.

It is possible for the number of Gaussians to increase exponentially over the time-steps. We prevent this by simply limiting the mixture size to a specific number. Of course there are better ways to address this. One alternative is to adaptively change the threshold ϵ_K, such that the total number of Gaussians in the mixture belief at the end of the horizon is computationally reasonable. However, this is beyond the scope of this work and is left as a motivation for future directions.

3.2 Gaussian Mixture Belief Reinforcement Learning

Having established a method to accurately propagate the belief forward in time, we can now formulate a belief space reinforcement learning algorithm. We define the augmented state $\mathbf{z}_k$ as the Gaussian mixture belief at time t_k. Formally,

$$\mathbf{z}_k \triangleq \begin{bmatrix} \boldsymbol{\mu}^1 \\ \text{vec}(\boldsymbol{\Sigma}^1) \\ \vdots \\ \boldsymbol{\mu}^{N_k} \\ \text{vec}(\boldsymbol{\Sigma}^{N_k}) \end{bmatrix} , \tag{3.8}$$

where N_k is the number of Gaussians that compose the mixture of the belief distribution at time t_k. Note that the dimensionality of the augmented state $\mathbf{z}$ may change within the same time-step, as a Gaussian can be split into a mixture before being propagated to the next time-step. This creates the problem in which the gradient and the hessians of V and Q with respect to the augmented state will also have to change dimensionality within the same time-step. To address this, we derive how to cast V and Q objects from one dimensionality to another.

We begin by considering the case wherein, at a particular time-step, our belief is a single Gaussian distribution $\mathcal{N}(\bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\Sigma}})$ and it is split into a mixture $\sum_{i=1}^{N} w^i \cdot \mathcal{N}(\boldsymbol{\mu}^i, \boldsymbol{\Sigma}^i)$. We have that

$$\mathcal{N}(\bar{\boldsymbol{\mu}}, \bar{\boldsymbol{\Sigma}}) \approx \sum_{i=1}^{N} w^i \cdot \mathcal{N}(\boldsymbol{\mu}^i, \boldsymbol{\Sigma}^i) , \tag{3.9}$$

due to our splitting method described in Section 3.1. And thus, by matching moments, we have that

$$\bar{\boldsymbol{\mu}} = \sum_{i=1}^{N} w^i \boldsymbol{\mu}^i \qquad \text{and} \qquad \bar{\boldsymbol{\Sigma}} = \sum_{i=1}^{N} w^i \left(\boldsymbol{\Sigma}^i + (\boldsymbol{\mu}^i - \bar{\boldsymbol{\mu}})(\boldsymbol{\mu}^i - \bar{\boldsymbol{\mu}})^\mathsf{T} \right) . \tag{3.10}$$

In the backwards pass of PDDP, we obtain the gradients and hessians of Q and V from the next time-step. As a result, we assume that we have these objects in the dimensionality of the Gaussian mixture and the objective is to cast them into the dimensionality of the original

single Gaussian belief. Formally, we have an object

$$A_{\mathbf{z}_{\text{GM}}} \triangleq \begin{bmatrix} \nabla_{\boldsymbol{\mu}^1} A \\ \nabla_{\boldsymbol{\Sigma}^1} A \\ \vdots \\ \nabla_{\boldsymbol{\mu}^{N_k}} A \\ \nabla_{\boldsymbol{\Sigma}^{N_k}} A \end{bmatrix}, \tag{3.11}$$

where $A \in \mathbb{R}$ representing V, Q, and its gradients, and $\mathbf{z}_{\text{GM}}$ is the belief in Gaussian mixture (GM) representation. Given this object, we must find

$$A_{\mathbf{z}} \triangleq \begin{bmatrix} \nabla_{\bar{\boldsymbol{\mu}}} A \\ \nabla_{\bar{\boldsymbol{\Sigma}}} A \end{bmatrix}, \tag{3.12}$$

where we adopted $\mathbf{z}$ simply as the original single Gaussian belief.

For the mean, we simply have that for each state dimension $a = 1, \ldots, d$

$$\frac{\partial A}{\partial \bar{\boldsymbol{\mu}}_a} = \sum_{i=1}^{N} \frac{\partial A}{\partial \boldsymbol{\mu}_a^i} \cdot \frac{\partial \boldsymbol{\mu}_a^i}{\partial \sum_j w^j \boldsymbol{\mu}_a^j} \tag{3.13}$$

$$= \sum_{i=1}^{N} \frac{\partial A}{\partial \boldsymbol{\mu}_a^i} \cdot \frac{1}{w^i}, \tag{3.14}$$

and hence,

$$\nabla_{\bar{\boldsymbol{\mu}}} A = \sum_{i=1}^{N} \nabla_{\boldsymbol{\mu}^i} A \cdot \frac{1}{w^i}. \tag{3.15}$$

As for the gradients with respect to the covariance, we obtain that for each state dimension pair $a, b = 1, \ldots, d$

$$[\nabla_{\bar{\boldsymbol{\Sigma}}} A]_{ab} = \frac{\partial A}{\partial \bar{\boldsymbol{\Sigma}}_{ab}} \tag{3.16}$$

$$= \sum_{i=1}^{N} \left[\sum_{c=1}^{d} \left[\frac{\partial A}{\partial \boldsymbol{\mu}_c^i} \cdot \frac{\partial \boldsymbol{\mu}_c^i}{\partial \bar{\boldsymbol{\Sigma}}_{ab}} + \sum_{e=1}^{d} \frac{\partial A}{\partial \boldsymbol{\Sigma}_{ce}^i} \cdot \frac{\partial \boldsymbol{\Sigma}_{ce}^i}{\partial \bar{\boldsymbol{\Sigma}}_{ab}} \right] \right]. \tag{3.17}$$

In particular,

$$\frac{\partial \boldsymbol{\mu}_c^i}{\partial \bar{\boldsymbol{\Sigma}}_{ab}} = (\delta_{ac} + \delta_{abc}) \frac{1}{w^i} (\boldsymbol{\mu}_c^i - \bar{\boldsymbol{\mu}}_c) \ , \tag{3.18}$$

where, δ is the kronecker delta so that $\delta_{a_1 \cdots a_n} = \begin{cases} 1, & \text{if } a_1 = \cdots = a_n \\ 0, & \text{otherwise} \end{cases}$. We also obtain

$$\frac{\partial \boldsymbol{\Sigma}_{ce}^i}{\partial \bar{\boldsymbol{\Sigma}}_{ab}} = \frac{\delta_{ac}\delta_{be}}{w^i} \ . \tag{3.19}$$

The expressions above fully describe how to incorporate Gaussian mixture belief into the PDDP framework.

Chapter 4

Results

In this thesis, we give emphasis on providing the theoretical derivation of our Gaussian mixture belief space RL algorithm. We only provide results for the forward belief propagation with GPs, however.

4.1 Gaussian Splitting

First, the accuracy of the Gaussian splitting method is evaluated. Fig. 4.1 describes the approximation of a univariate Gaussian distribution with a mixture of Gaussians. We show that the approximation is accurate, such that the original and the approximation are indistinguishable, and the ISD value is negligible.

4.2 Moment matching

In addition, we show that the moment matching propagation with a Gaussian mixture is more accurate than with single-Gaussian belief. Fig. 4.2 shows how a Gaussian mixture can easily handle multimodality in the true output distribution. The red line in the left box can accurately approximate the samples from the true distribution shown by the blue histogram.

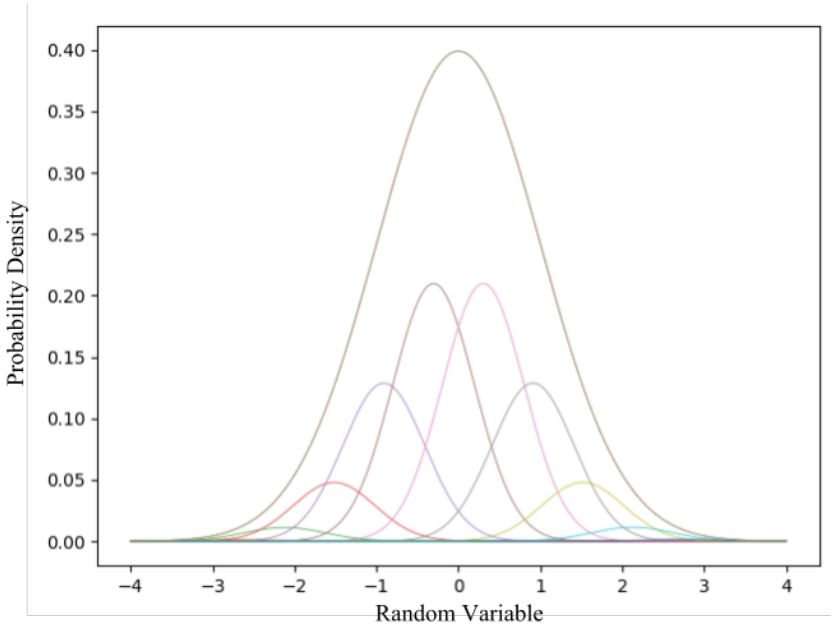

Figure 4.1: **Gaussian splitting.** Central (blue) curve describes the original Gaussian. Smaller curves are the individual weighted Gaussians in the mixture. Central (orange) curve describes the final Gaussian mixture. Note that the original and the approximation distributions are visually indistinguishable.

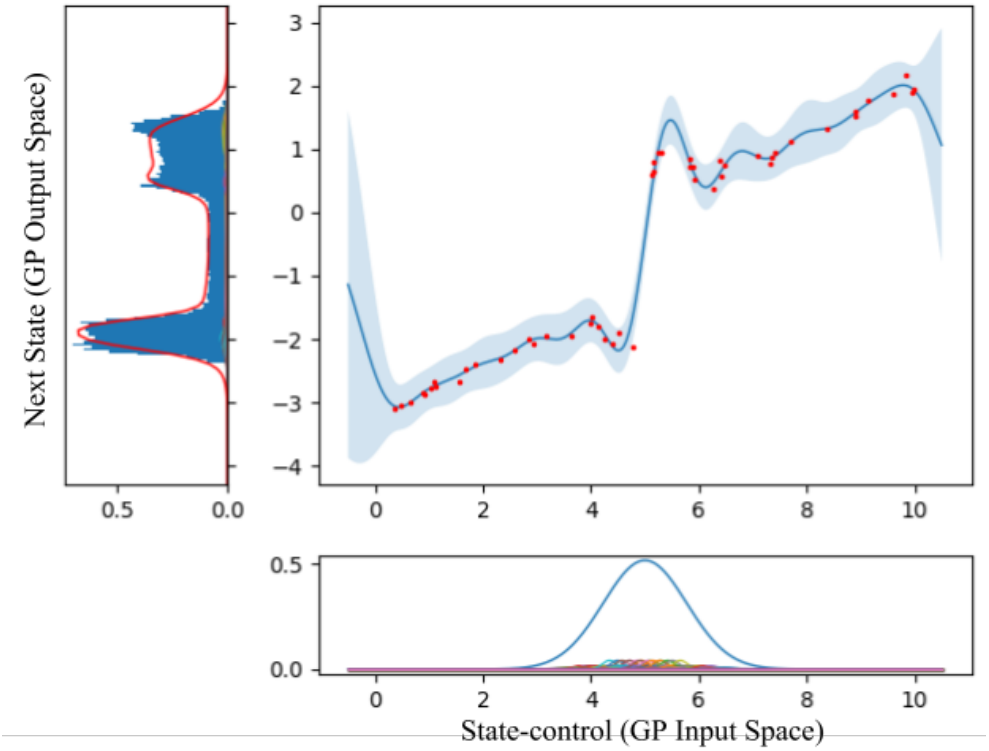

Figure 4.2: **Propagating Gaussian mixture over one time-step.** (Bottom) Input Gaussian belief in blue and Gaussian mixture approximation in the remaining colors. (Middle) Gaussian process: data points in red; mean function in strong blue line; ± 1 standard deviation in light shaded blue. (Left) Output distribution: sampled (true) distribution in shaded dark blue; propagated Gaussian mixture belief in red.

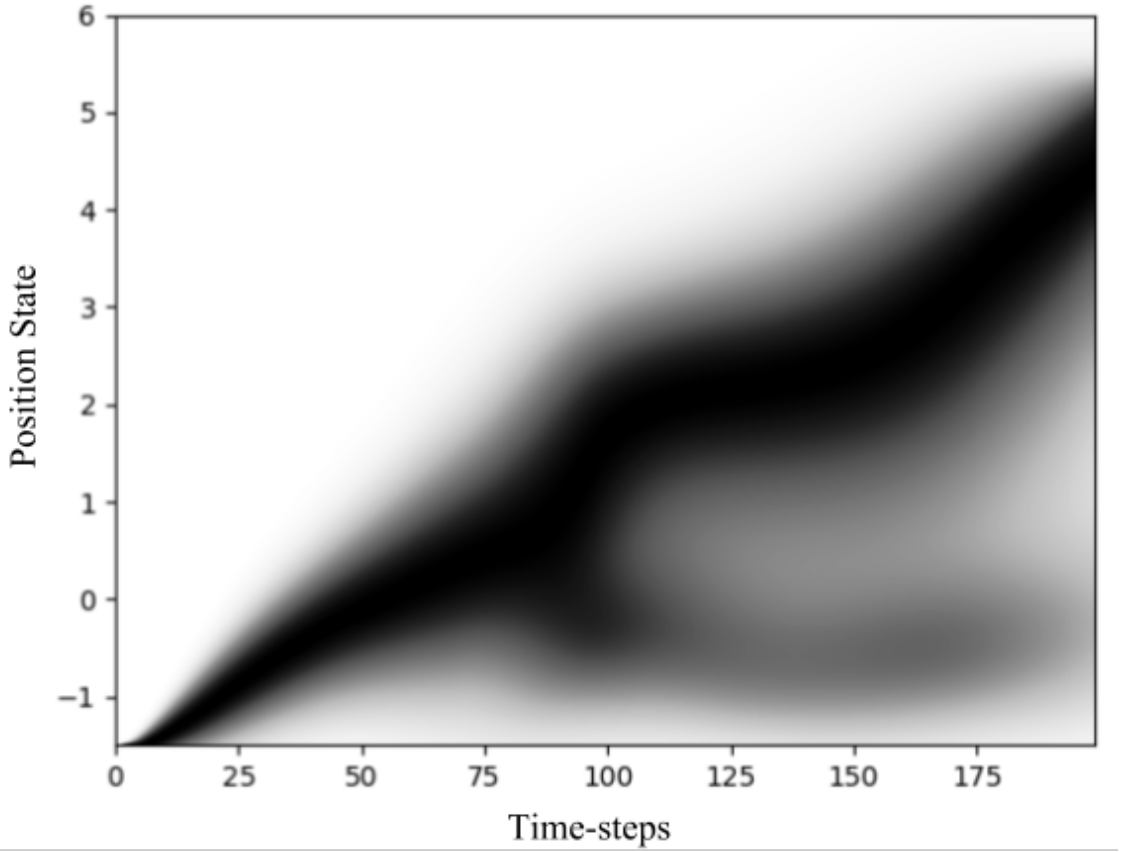

Figure 4.3: **Propagating Gaussian mixture over multiple time-steps.** At each time-step (vertical slice) the state belief is represented by the shade of black. The darker the shade, the higher the density at that particular state in the vertical axis.

We also show the belief propagation over multiple time-steps in a stochastic system that contains strong bi-modality. In Fig. 4.2, we show how the belief over the states (in the vertical axis) propagates over time (in the horizontal axis). As seen in the figure, the method used can easily capture the bi-modality of the stochastic system. A single-Gaussian approximation would have unreasonably large variance.

Chapter 5

Conclusion

In this work, we have presented a data-driven, sample-efficient, probabilistic model-based reinforcement learning method for systems with unknown dynamics with multi-modal stochasticity. Our method, named Gaussian Mixture Differential Dynamic Programming (GaMix-DDP), is an extension of Probabilistic Differential Dynamic Programming (PDDP) [Pan and Theodorou, 2014] that can only handle a single Gaussian belief at every time-step. GaMix-DDP is able to handle stochasticity and uncertainty of the dynamics by learning the model with a Gaussian process, thus explicitly representing such uncertainty. Using a mixture of Gaussian beliefs at each instant in time, our method extends PDDP by representing the state belief with multi-modal distributions and thus more accurately approximating the true distribution of state trajectories. As in PDDP, GaMix-DDP considers a quadratic approximation of the value function and locally optimizes for a linear control policy. By learning the dynamics with GPs, GaMix-DDP yields a highly data-efficient control algorithm, wherein very few dynamics samples are needed to learn a model and optimize for the control.

We have shown that our method can approximate the true distribution of trajectories more accurately, which allows our algorithm to converge to the locally optimal control policy significantly faster. Finally, with more freedom to represent the state belief, our method yields better performance, as compared to PDDP. Our method, being very sample-efficient and achieving high performance, offers many opportunities for robotics applications.

5.1 Future Work

Although GPs are very data-efficient, the method used to propagate the state belief forward in time with matrix inversions and analytic moment matching is very computationally expensive. As a result, more computationally efficient methods can be investigated to be incorporated into this probabilistic control framework. For instance, methods such as dropout Bayesian neural networks [Gal and Ghahramani, 2016] and ensemble of networks [Lakshminarayanan et al., 2017] have shown to model epistemic and aleatoric uncertainty well, while still being computationally cheap. The challenge for these other uncertainty representation and propagation methods, however, is that they are not as data-efficient as GPs, due to their parametric nature. Addressing these challenges are left for future work.

Moreover, this method for multi-modal belief propagation, developed by Hardy et al. [2015], can also be used to extend control algorithms based on the path integral framework [Theodorou et al., 2010]. Pan et al. [2015] has presented a path integral control approach using single Gaussian belief, which can also be extended to using Gaussian mixtures to represent the state belief.

Finally, this work can be applied to any reinforcement learning algorithm that models dynamics in a probabilistic manner. For instance, Buckman et al. [2018] used ensemble networks to learn the dynamics and expand the value function for more accurate Q-function targets. In this setting, our method can be used to expand the value function even more accurately.

Grounded on the extensive analysis of classical trajectory optimization and Bayesian machine learning, this work offers a framework that can extend various decision-making and control algorithms.

References

Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, 2nd edition, 2000. ISBN 1886529094.

Jacob Buckman, Danijar Hafner, George Tucker, Eugene Brevdo, and Honglak Lee. Sample-efficient reinforcement learning with stochastic ensemble value expansion. 2018. URL `https://arxiv.org/pdf/1807.01675.pdf`.

Marc Peter Deisenroth. Efficient reinforcement learning using gaussian processes. 2010.

Marc Peter Deisenroth, Dieter Fox, and Carl Edward Rasmussen. Gaussian processes for data-efficient learning in robotics and control. *IEEE Trans. Pattern Anal. Mach. Intell.*, 37(2):408–423, 2015. URL `http://dblp.uni-trier.de/db/journals/pami/pami37.htmlDeisenrothFR15`.

MP. Deisenroth and CE. Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on Machine Learning, ICML 2011*, pages 465–472. Omnipress, 2011.

Yarin Gal and Zoubin Ghahramani. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 1050–1059. JMLR.org, 2016. URL `http://dl.acm.org/citation.cfm?id=3045390.3045502`.

Jason Hardy, Frank Havlak, and Mark Campbell. Multi-step prediction of nonlinear gaussian process dynamics models with adaptive gaussian mixtures. *The International Journal of Robotics Research*, 34(9):1211–1227, 2015. doi: 10.1177/0278364915584007. URL https://doi.org/10.1177/0278364915584007.

Frank Havlak and Mark E. Campbell. Discrete and continuous, probabilistic anticipation for autonomous robots in urban environments. *CoRR*, abs/1309.0766, 2013. URL http://arxiv.org/abs/1309.0766.

David H. Jacobson and David Q. Mayne. *Differential dynamic programming [by] David H. Jacobson [and] David Q. Mayne*. American Elsevier Pub. Co New York, 1970.

Balaji Lakshminarayanan, Alexander Pritzel, and Charles Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6402–6413. Curran Associates, Inc., 2017. URL http://papers.nips.cc/paper/7219-simple-and-scalable-predictive-uncertainty-estimation

Yunpeng Pan and Evangelos Theodorou. Probabilistic differential dynamic programming. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 1907–1915. Curran Associates, Inc., 2014. URL http://papers.nips.cc/paper/5248-probabilistic-differential-dynamic-programming.pdf.

Yunpeng Pan, Evangelos Theodorou, and Michail Kontitsis. Sample efficient path integral control under uncertainty. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2314–2322. Curran Associates, Inc., 2015. URL http://papers.nips.cc/paper/5984-sample-efficient-path-integral-control-under-uncertai

Robert Platt, Leslie Kaelbling, Tomas Lozano-Perez, and Russ Tedrake. Non-gaussian belief space planning: Correctness and complexity. In *IEEE Conference on Robotics and Automation (ICRA)*, 2012. URL http://lis.csail.mit.edu/pubs/tlp/ICRA12$_1$775$_F$*I.pdf*.

Robert Platt, Leslie Kaelbling, Tomas Lozano-Perez, and Russ Tedrake. *Efficient Planning in Non-Gaussian Belief Spaces and Its Application to Robot Grasping*, pages 253–269. Springer International Publishing, Cham, 2017. ISBN 978-3-319-29363-9. doi: 10.1007/978-3-319-29363-9$_1$5.URL